A Novel Algorithm for Flow-Rule Placement in SDN Switches

Kentis, Angelos Mimidis; Pilimon, Artur; Soler, José; Berger, Michael Stübert; Ruepp, Sarah Renée

Published in:
Proceedings of the 4th IEEE International Conference on Network Softwarization

Link to article, DOI:
10.1109/NETSOFT.2018.8459979

Publication date:
2018

Document Version
Peer reviewed version

A Novel Algorithm for Flow-Rule Placement in SDN Switches

Angelos Mimidis-Kentis*, Artur Pilimon*, Jose Soler, Michael Berger and Sarah Ruepp
Department of Photonics Engineering
Technical University of Denmark
Lyngby, Denmark
{agmimi, artpil, joss, msbe, srru}@fotonik.dtu.dk

***Abstract*— The forwarding rules, used by the legacy and SDN network devices to perform routing/forwarding decisions, are generally stored in Ternary Content Addressable Memory (TCAM) modules, which offer constant look-up times, but have limited capacity, due to their high capital and operational costs, high power consumption and high silicon footprint. To counter this limitation, some commercial switches offer both, hardware and software flow table implementations, termed hybrid flow table architecture in this paper. The software-based tables are stored in non-TCAM memory modules, which offer higher capacity, but with slower lookup times. In addition, these memory modules are limited in terms of how many requests they can serve per time unit. Thus, exceeding this threshold will lead to packet loss in the network. This paper proposes a novel placement algorithm, which dynamically decides whether a new flow rule should be placed in a hardware (expensive) or a software (cheap) table. The placement decisions are based on a number of criteria with the goal to increase the utilization of the software-based table, without introducing performance degradation in the network in terms of significant delay and packet loss. The performance of the placement algorithm was evaluated through experimental measurements in a testbed, which comprises a hybrid SDN switch, a server performing traffic generation and a server hosting the SDN controller. The results indicate that, by limiting the maximum allowed processing capacity of the software table, the number of accommodated flows is significantly increased, while bounding any excessive delays and avoiding packet loss.**

Keywords— SDN, Flow tables, TCAM, OpenFlow Pipelines

I. INTRODUCTION

Regardless of the applied control plane paradigm (centralized or distributed), network devices make routing decisions based on rules that reside within one or more device-local rule tables. Whenever a packet arrives in a network/forwarding device, its headers are cross-checked against the rules that populate those tables and depending on the result (match or no match) one or more actions might be applied to the packet by the device (e.g. forward, drop). The granularity with which the packets are examined (i.e. the number of headers checked), and the implementation and/or number of tables involved, can vary between different devices and networking paradigms.

Given the need for fast packet processing, most physical network devices implement their forwarding tables using Ternary Content Addressable Memory (TCAM) modules, which provide O(1) lookup times (in terms of clock cycles) [1]. That means that regardless of the number of entries in the table(s), finding a match (or not) will always take the same amount of time. This trait is highly desirable; since apart from generally fast search times, it also provides a high level of determinism (all look-ups take always the same time). However, TCAM modules are expensive, have high power consumption and a large silicon footprint [1]. In order to cut-down on the associated Capital and Operational Expenditures (CAPEX, OPEX), network device vendors limit the size of the TCAM modules, which results in reduced number of flow-rules to be stored [1].

The Software Defined Networking (SDN) paradigm offers a greater granularity for defining flow-rules (more packet headers can be defined as match fields), when compared to the traditional network paradigm. This approach provides increased flexibility when designing applications for SDN Controllers (SDNCs), but requires more memory space per flow rule. Given the limited size of the flow tables, this characteristic can limit the applicability of SDN in network deployments with extensive number of flows. To resolve this issue, both SDN-enabled device vendors and SDNC application developers (from the industry and the academia) have attempted to increase the effective capacity of the flow tables. With regards to vendors the approach is to provide an additional flow table implementation in their devices based on software. This can offer increased flow-rule capacity by sacrificing search performance (exceeding O(1)). On the other hand, SDNC developers have mostly focused on developing flow-rule aggregation mechanisms, which allow the network devices to process more traffic with the same number of flow-rules. The drawback of this approach is that aggregation of network flows reduces the processing granularity offered by SDN. These approaches are covered in more detail in the related work section of this paper.

This work builds on the software table implementations, by proposing a dynamic and intelligent flow-rule placement algorithm, executed in the SDNC. The target is to maximize the number of flow-rules residing in a switch, while also

limiting the negative effects (e.g. increased delay, packet loss), imposed by the memory modules (hardware or software).

The remainder of this paper is structured as follows. Section II provides an overview of the related research work based on flow-rule placement and flow aggregation algorithms. Section III discusses the available switch architectures, focusing on their flow table implementations (pure hardware, pure software and hybrid). Section III discusses flow table pipeline implementations and how they can affect the performance of flow-rule placement algorithms. Section IV presents the proposed flow placement algorithm for hybrid flow table architectures. Section V describes the experiments conducted to evaluate the proposed algorithm, together with the collected results. This paper is concluded and possible future steps are outlined in Section VI.

II. Related Work

Efficiency of the flow rule installation process in SDN switches has been a matter of high research interest in the SDN community. Different approaches to manage the TCAM space utilization have been proposed and evaluated, primarily targeting the flow rule compression or aggregation [2 – 5], or flow rule caching and placement algorithms [6 – 12]. Hence, we further provide a summarized yet comprehensive overview of the relevant literature findings.

A. Flow rule aggregation

Flow rule aggregation aims at reducing the flow table size in the network nodes by substituting a set of rules with overlapping matching criteria with a more generalized flow rule, while still being able to realize a corresponding network policy. The variants of this procedure are commonly referred to as traffic flow aggregation [2][3] and flow table compression [4][5]. An approach for dynamic flow aggregation was proposed in [2], where a dynamic traffic aggregation decision is made based on two criteria: the computed network path of the flows and their DSCP (Differentiated Services Code Point) marks. The traffic flow aggregates are identified by adding unique per-flow-aggregate VLAN (Virtual Local Area Network) tags. However, the performance of the aggregation service was measured in a simulated network with software switches (on a single physical server) and a limited number of traffic flows. Rifai *et* al. in [4] proposed a framework, called MINNIE, for flow table compression using wildcard rules and shortest-path routing using adaptive metrics (link, router load) for load balancing. The compression mechanism produces a set of three tables, using compression by source, by destination and by default flow rule. The smallest resulting compressed table is chosen for the routing decisions. Experimental measurements, described in [4], were conducted , on a testbed, containing a commercial SDN switch with a hybrid (software-hardware) flow table design. The results show that even when using this compression technique, the first packet (of each flow) delay increases by a factor of up to 20 and the average matching delay for the remaining packets results in 6-fold increase when installing the flow rules in software as compared to hardware (TCAM). An incremental flow table aggregation mechanism is discussed in [5], where the authors propose a set of two algorithms, namely FFTA (Fast Flow Table Aggregation) scheme applied to non-prefix (TCAM-based) flow rules. An offline version of FFTA is used for initial partitioning of the flow rules, applying prefix aggregation and then merging together the rules with a single differing bit in an iterative manner. The online version allows performing fast incremental rule updates with a small loss of compression ratio.

B. Flow rule placement algorithms

This category of flow rule distribution methods includes flow rule caching and rule placement in general. In [6][7] authors present a design of a hybrid hardware-software switch abstraction with arbitrarily large flow rule tables. This is achieved by using a complex rule caching mechanism, consisting of a rule placement algorithm, called CacheFlow, a Cache master module and a set of elastic shared software switches. The rule placement algorithm constructs a rule dependency tree and caches the most popular flow rules (serving a large volume of "cache hit" traffic) in the TCAM, but redirects smaller amount of "cache miss" traffic to be handled by the software switches. If there is no matching rule found either, the controller is contacted as the last instance for a new flow rule installation. This system allows achieving several important goals: a) avoid cache replacement without taking into consideration possible complex flow rule dependencies (pattern overlaps); b) avoid flow compression to preserve the OpenFlow semantics, i.e., per-flow-rule traffic counts; c) reduce the size of the long chains of dependent rules by "splicing" such chains to cache smaller groups of rules [7]. Another flow rule caching optimization method, called CRAFT is introduced in [8]. This mechanism uses a two-stage pipeline to eliminate the need for slow processing of long rule dependency chains, to reduce the possibility of having overlapping flow rules in the space-limited cache. The cache expansion problem is solved by weighted splitting of large flow rules into sub-rules and only caching the sub-rules with the highest weight (hit ratio). This scheme is reported to be 30% more efficient as compared to the CacheFlow [7].

Guo *et* al. in [9] propose a novel traffic forwarding scheme coupled with reactive flow rule placement, called JumpFlow. The forwarding module of the algorithm uses the VLAN identifier (VID) field of the packet header to carry the routing information, while the rule placement module divides the complete flow's forwarding information into several blocks and installs them on a selected subset of contact switches (along the flow's path). The objective of the reactive module is to maintain low and balanced flow table (TCAM) utilization by applying constraints of flow table space and the number of contact switches to use, with an optimal solution achieved using Integer Linear Programming (ILP).

In [10], the authors employ a flow rule partition and allocation strategy, where the flow rules in heterogeneous flow tables are split and grouped into sub-tables (stored in a virtual small TCAM block), which are then distributed across the entire network as uniformly as possible. Only the hardware (TCAM) flow tables are targeted. The main goal of this

approach is to divide all flow rules into disjoint sub-tables, putting the rules that implement the same policy or have dependency in the same sub-tables.

A novel solution to optimize the TCAM memory usage is proposed in [11], by implementing a Memory Management System (MMS) component for the SDNC. It performs memory swapping by temporarily moving the least used flow rules from the TCAM space to the external database (residing in the MMS of the SDNC). Then, when the load of the TCAM table decreases, the MMS automatically restores the swapped-out rules upon demand (e.g., a new packet matching one of these rules arrives).

Other solutions, e.g., as in [12], focus on flow-driven rule caching optimization, where authors achieve a high cache hit ratio by prefetching (caching over all the switches along a flow's path) the flow rules that need to be cached for fast path processing and setting a timer with an estimated time of the next rule "hit" event. This is achieved by analyzing the routing paths of each flow and its detected traffic pattern.

Our proposed rule placement approach differs from related work in several ways, even though some conceptual similarities with the discussed works are present. First, we are not targeting the flow table compression to retain the possibility to obtain per-flow-rule traffic statistics and avoid introducing any need for recalculating the optimal number of compressed rules, which can be an NP-hard task, since we are considering a dynamic reactive flow rule migration. Second, unlike in the case of a CacheFlow [6][7] approach, where additional processing overhead is introduced by embedding the software switches (with additional pipeline processing) and extra coordination component (cache master), we utilize the properties of the hardware and software tables and keep the main algorithmic logic in the SDNC. Third, we are not modifying the packet headers to perform flow grouping by similar properties (e. g., packet rates), but instead we are using a predefined mapping of flow group rates to transport protocol destination ports. Finally, our work is conceptually related to [11], since we are swapping the flow rules between different memory types, but we retain this process within the memory space of the switch, rather than exporting the rules externally that incurs varying delays.

III. SWITCH ARCHITECTURES

When evaluating the performance and utilization of flow table implementations, there are three architectural components that must be taken into consideration. These are:

- How the flow tables are implemented (pure hardware, pure software or hybrid)
- How the flow tables within a single device are interconnected; a mechanism referred to as the *packet processing pipeline* of the device.
- How flow rules are allocated between the different flow tables; a mechanism referred to as a *flow rule placement algorithm*.

A. Flow table Architectures

There are three means to implement flow table architectures. They can be realized using pure hardware resources (e.g. TCAM), pure software resources or in a hybrid combination of these two, where some tables are implemented in hardware and some in software.

Pure software flow table implementations are almost exclusively encountered in virtual switches (e.g. OpenvSwitch [13]). Virtual switches are mostly used in Data Center (DC) environments, to forward traffic between virtual machines or containers, which reside within a single physical node. Because of their locality these switches handle only a limited number of traffic flows, hence the associated look-up operations do not impose significant performance degradation. Pure hardware implementations are mostly found in physical, legacy (non-SDN) network devices. Hybrid implementations are a relatively new approach, most commonly found in SDN-enabled network devices. This is because, through the programmability offered by the SDNC, dynamic flow rule placement algorithms can be implemented and enforced in the network infrastructure.

As summarized in Table 1, each flow table implementation has several benefits and drawbacks. Pure hardware implementations offer a fast (and constant) per packet look-up and also a high processing capacity, meaning they can handle traffic of high packet rates. However, their flow table capacity is limited, due to the CAPEX and OPEX costs associated with the TCAM modules. Pure software implementations on the other hand, offer a much higher flow table capacity, but at the cost of slow look-up and low processing capacity. Since they do not require a flow table placement algorithm, both hardware and software implementations have a relatively low complexity. Finally, hybrid implementations (if correctly utilized) can offer high flow table and processing capacities, as well as fast look-ups. The only disadvantage is the need for a placement algorithm, which can increase implementation complexity. However, in this paper we argue that if the placement algorithm is of a simple and efficient design, the benefits can out-weight the introduced complexity.

Table 1: Comparison of flow table implementations

Type of Implementation	Flow table Capacity	Lookup Speed	Processing Capacity	Complexity
Pure Hardware	low	high	high	low
Pure Software	high	low	low	low
Hybrid	high	high	high	high

B. Packet Processing Pipelines

Within the context of the SDN paradigm, a packet processing pipeline refers to the logic of the internal packet processing within a network device. There are two approaches for designing flow table pipelines, using a single flow table in which to store all flow rules or using multiple interconnected tables and store rules in them based on a set of criteria. The approach is dependent on both the underlying capabilities of the network device, but also on the protocol used for the control plane between the SDNC and the device (e.g.

OpenFlow 1.0 does not support multi-table pipelines but OpenFlow 1.3 does). When considering a single table pipeline, all incoming packets in the network device are cross-checked against this table. In case there is a match, the packet is processed based on the actions associated with the matching rule; if there is no match then the packet is sent to the SDNC. In a multi-table pipeline, flow processing can be composed of multiple flow rules, spread across the different tables. This means that an incoming packet can be processed by multiple tables, allowing for more complex action sets to be enforced. Using a single flow table offers a lower implementation complexity, but using multiple flow tables allows for more efficient and dynamic flow table utilization. Figure 1 illustrates the two pipeline approaches.

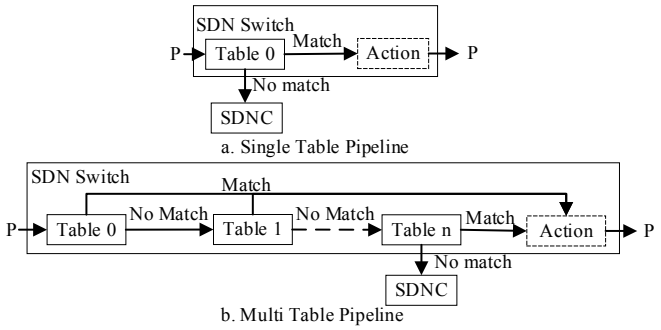

Figure 1: Pipeline models

IV. Proposed Pipeline and Placement Algorithm

As mentioned, hardware flow tables offer high and constant service rates (packets per second they can process), but are limited in the number of flow rules they can accommodate. On the other hand, software flow tables can accommodate more flow rules but have limited service rates. Additionally, for software tables the time it takes to service a request is directly related to the current number of flow rules in the table, which implies that their performance deteriorates as the number of flow rules, present in the software table, increases. This section presents the design and logic of the proposed flow rule placement algorithm and the selected packet processing pipeline. Both the placement algorithm and the pipeline, were implemented with the aforementioned benefits and drawbacks in mind.

A. Packet Processing Pipeline

In the SDN paradigm and with OpenFlow [14] as the control plane protocol, when a switch receives a packet, it cross-checks it against its flow rules for a match and then applies the associated actions to it. The outcome, however, is dependent not only on the defined action set, but also on how the pipeline processing within the switch is implemented. In this work, it was decided to process the incoming packets first at the hardware table and then the software table. This approach removes unnecessary processing stress from the software table as it is only accessed when a match is not found in the hardware table. If neither the hardware of software table holds a matching flow rule, then the switch will ask the SDNC for further instructions with an OpenFlow PacketIn message. Upon receiving the PacketIn message, the SDNC will process it and decide how the packet should be treated in the network (f. ex. forwarded, dropped, modified etc). The means through which, the SDNC processes the request and decides on the packet treatment is out of the scope of this paper. After the packet treatment has been decided by the SDNC and before it is enforced in the network (by means of OpenFlow FlowMod messages), the proposed flow rule placement algorithm takes place. The implemented pipeline is illustrated in Figure 2.

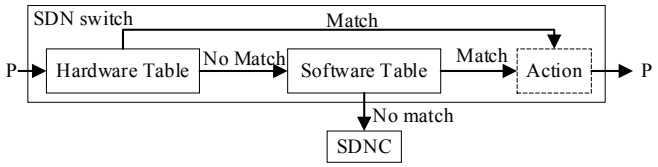

Figure 2: Proposed pipeline processing

B. Flow Rule Placement Algorithm

Upon receiving a request from a switch, the SDNC will query a local statistics database and retrieve the number of flow entries that populate the switch's hardware flow table. The information contained in the database is collected by means of a polling mechanism, which periodically retrieves switch related statistics. Even though the SDNC is the only entity managing the network, it is not safe to assume that it can keep track of the number of active flow entries in each switch without such a polling mechanism. This is because some of the flow rules in the switches might expire and be removed without the SDNC being notified (e.g., the SDN switch might not send an OpenFlow FlowRemoved message to the SDNC). By setting a high polling frequency, the accuracy of the collected statistics can be set to acceptable standards, at the expense of extra overhead in the interface between the SDNC and the network infrastructure.

Upon retrieving the number of flow rules in the hardware table, the algorithm compares the value against a predefined threshold. If the number of flow rules is below this threshold, then there is no imminent danger of overflowing the hardware table. Since the performance of the hardware table is superior to the software table, the flow rule is added to the hardware table. If the number of flow rules is greater than the defined threshold, then there is a danger that inserting the flow rule in the hardware table will cause a table overflow and disrupt network connectivity (e.g., packet drops, SDN switch crash). To mitigate this danger, the placement algorithm triggers a flow rule migration process from the hardware table to the software table. There are two elements of this migration process that need to be addressed here, namely how many flow rules are migrated each time the process is triggered, and which flows are selected for migration. Based on the issues addressed above, Figure 3 illustrates the proposed algorithm in the form of a flow chart. Table 2 lists the different variables used in the chart.

With regards to how many flows to migrate, three approaches have been identified. The algorithm could migrate one flow, migrate K flows (K could be either a static predefined value or dynamic based on the current situation), or

finally the algorithm could migrate as many flows as possible until the service rate of the software table is saturated. Each approach has its own benefits and drawbacks which are presented below.

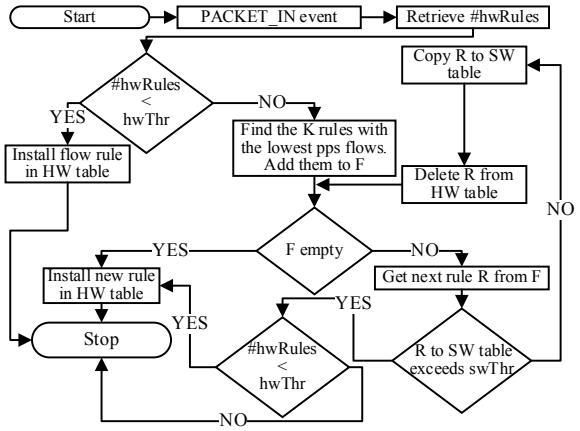

Figure 3: Flow chart of the flow rule placement algorithm

Table 2: List of flow chart variables

Variable	Description
#hwRules	The number of flow rules existing in the hardware table of the switch.
hwThreshold	The threshold, expressed as number of flow rules, which identifies a critical point after which the hardware table is prone to table overflow.
swThreshold	The threshold, expressed as packets per second, which identifies a critical point after which the software table can become unresponsive.
F	A list which holds all flow rules considered for migration.
R	A single flow rule considered for migration
K	The static or dynamic value, denoting how many flow rules to consider for migration on every iteration.
pps	Packet per second rate of a flow.

Migrating as many flow rules as possible, reduces the instances in which the threshold is reached, hence limiting the number of times the migration process is initiated. However, this approach always leads to the full utilization of the software table, which will hinder network performance due to increased delays for the migrated flows. In contrast, migrating just one flow rule whenever the algorithm is triggered minimizes the utilization of the software table. However, unless some flow rules from the hardware table expire or are removed by the SDNC, this approach requires one iteration of the placement algorithm for each new flow arriving at the switch. This makes the algorithm more computationally expensive, as well as it increases the response time of the SDNC to service requests from the network. The final approach and the one selected for this work, is to migrate K flow rules per iteration of the flow rule placement algorithm. Doing so provides the benefits of both previous scenarios, since the number of times the algorithm is triggered is limited but so is the utilization of the software table. It is important to stress that independent of the selected approach (1 flow rule, K flow rules, max flow rules), a flow rule should be migrated to the software table if and only if, the resulting cumulative packet per second rate of the software table is under a predefined threshold. Exceeding this threshold means exceeding the processing capabilities of the software table, leading likely to disruptions in network connectivity. If that is the case and the hardware table can accommodate the flow, it will be added there. Else the flow will be dropped (neither the hardware nor the software table can accommodate it).

Based on the correlation between service requests (packets per second) and service times (time it takes to find a matching flow rule) for software tables, the proposed algorithm selects the K flows with the lowest packet rate for migration. This way the utilization of the software table is kept to a minimum. Most SDNCs, offer flow level statistics which include per flow packet rates, however, the accuracy of these statistics is very coarse as they are based on periodic polling with an interval at the order of seconds. Given that the packet rate of a flow can vary significantly during its lifetime, using these statistics can lead to incorrect assumptions on the flow's packet rate. Migrating a flow rule based on a wrongly assumed packet rate can lead to over provisioning of the software table which, in turn, can lead to either packet losses or excessive delays. The means, through which the packet rates of flows are identified, are out of the scope of this paper. However, some possible solutions are either the increase of the polling frequency from the SDNC to sub-second values or the use of network analytics techniques (e.g., sFlow). For the proof of concept implementation of the algorithm, the packet rates of each flow are considered constant and are also identifiable from the SDNC by packet header values, where each destination UDP port implies a specific packet rate.

To avoid network connectivity disruptions for the flows of the migrated flow rules, a migrate-then-delete approach was selected. Since the hardware table resides first in the pipeline processing, this model assures that there will always be at least one active copy of the flow rule within the switch. The drawback of this approach is that temporarily there will be two identical flow rules in the switch, one on each flow table. However, this only holds true for a very limited amount of time, since the migration process is executed relatively fast. Figure 4 illustrates an example of a flow rule migration instance for K = 3.

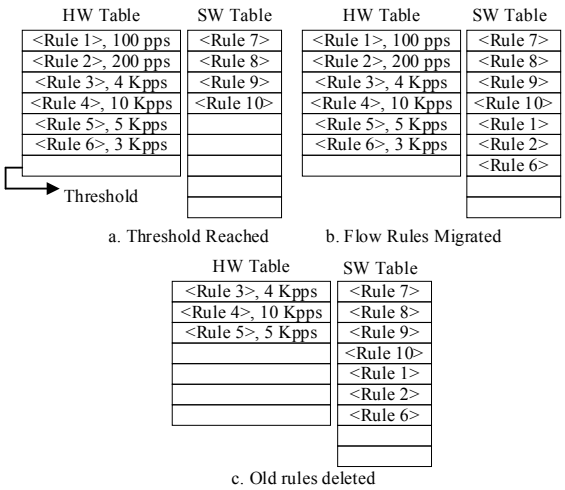

Figure 4: Example of the flow rule migration process, with K = 3

The flow rules 1, 2 and 6 are migrated since their corresponding flows have the lowest packet rates. In (a), the SDNC identifies that the threshold for the hardware table has been reached so it will initiate the migration process. In (b) the flow rules 1, 2 and 6 are migrated to the software table and in (c) they are deleted from the hardware table.

The last element that needs to be addressed is how the threshold values for the hardware (number of rules) and software (packets per second) tables are set by the algorithm. Defining the threshold for the software table is straightforward, since the packet service capacity of the software table is known from the device's datasheet and is independent of any variables (e.g. packet size). Defining a threshold for the hardware table on the other hand is a more complex task. This is because the number of rules that a hardware table can accommodate is not a static value but can vary depending on how coarse/granular each installed flow rule is. The more header fields are defined for matching in a flow rule, the more space this flow rule occupies in the table. Based on this observation, there are two approaches that can provide a secure threshold value. The first is to calculate exactly how many bytes each flow rule occupies and then sum all the values together; the sum can then be compared against the total space provided by the hardware table. However, this approach implies knowledge of how much space each unique header field will occupy, information not necessarily available to the SDNC. Another approach, and the one selected for the PoC implementation, is to follow a worst-case scenario in which it is assumed that all flow entries occupy the same amount of space, equal to the case in which all header fields are defined for matching. This approach has the obvious drawback of limiting the effective size of the hardware table, but due to its simplicity it was selected for the PoC. As a future step, a more robust mechanism for calculating the available space should be implemented.

V. Validation and Results

To validate the functionality of the implemented PoC algorithm, a set of experiments was conducted on a physical SDN testbed. The testbed comprised a server for generating and receiving traffic flows, a physical SDN switch with hybrid flow table implementation and finally a server hosting the SDNC [15] in which the flow rule placement algorithm was executing. The server, responsible for generating the traffic flows, was equipped with 2-port NIC with one port for transmitting and one for receiving the traffic flows. Both NIC's ports were then connected to the SDN switch. The reason for using a single server for sending and receiving traffic flows was the need for a common reference clock for the delay measurements. Finally, the SDN switch was connected to the SDNC though the management interface. Figure 5 illustrates the testbed setup that was used.

The scope of the presented experiments is threefold. First, to validate that the flow rule placement algorithm works as intended by migrating flow rules from the hardware to the software table, based on the defined threshold values. Second, to evaluate, if the algorithm introduces any performance degradation in the network, when compared to the default scenario, where all flows are placed in the hardware table. Third, to observe the combined impact of higher flow pps rates when migration is activated, while keeping the packet loss as low as possible to ensure accurate latency measurements. Due to the limitations imposed by the traffic generation software, it was not possible to saturate the capacities of the hardware and software tables. To mitigate this issue, the SDNC was utilized to "virtually" cap the capacities of both tables to lower values. For the hardware table the limit was set to 99 flow rules and for the software table to either 400 or 600 packets per second (pps) depending on the experiment, defined as follows.

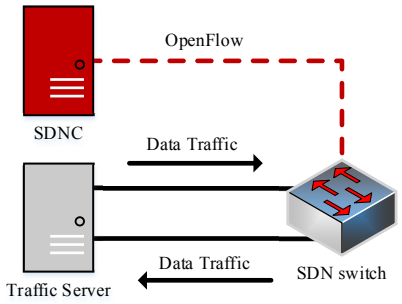

Figure 5: Testbed

The following traffic generation experiments were designed with the scope of stressing the (capped) capacities of both the hardware and software tables. There are two experiments with 150 unique traffic flows in each, with the flows evenly spread amongst three packet rate groups. In the first scenario, there are 50 flows with 10 pps, 50 with 20 pps and 50 with 30 pps. The second scenario is comprised of 50 flows with 15 pps, 50 with 30 pps and 50 with 45 pps. The capacity of the software table was limited to the 400 pps for the first scenario and to 600 pps for the second scenario, with the intent to be able to reach the overflow state for the software table in both scenarios. The traffic flows were sequentially generated in a round robin fashion from each packet rate group within each scenario. Execution of the experiments resulted in the expected behavior. Initially, all flow rules were installed in the hardware table, however when the hardware threshold was reached (set as 95% of the capacity), then the migration process was initiated, and a set of flows to migrate from the hardware to the software table was iteratively being chosen. This process was repeated until the processing capacity of the software table was saturated, and the migration process stopped. After this point the hardware table utilization reached 100% of capacity, and all subsequent flows were rejected.

To evaluate the performance of the algorithm the same experiments as before were repeated with and without the placement algorithm enabled. In the first set of experiments, which will be referred to as *baseline scenario*, only the hardware table is used to serve the arriving flow processing requests. The second set of experiments will be referred to as *flow migration scenario* and are used to benchmark the

performance (in terms of delay) of the flow rule placement algorithm.

It is important to note that the timescales (on the horizontal axes) of the graphs, presented further, are relative per-flow timescales, rather than on a single universal timescale for all the flows. Hence, the last plotted value on any per-flow timescale denotes the total flow duration in seconds. However, this aspect does not affect our analysis, since we are not plotting the delays of groups of flows in a single graph as a function of time.

The results for the *baseline scenario* are presented in Figure 6 and Figure 7. As it can be seen in Figure 6, the distribution of the average per-flow delay for both packet rate sets (10-20-30 pps and 15-30-45 pps) in the baseline scenario is very similar to a uniform pattern with the mean value around 0.175 ms for the first set, 0.179 ms for the second set, and the average maximum delay not exceeding 0.2 ms. Such performance is expected, because the flow rules are placed only in the hardware table (which offers constant lookup times); if there is no remaining space to accommodate a new flow, the packets of that flow will be dropped. This is the behavior illustrated in Figure 6, where there are 99 accommodated flows (out of 150), adhering to the virtually imposed hardware table capacity limit (99 flow rules).

In addition, as it can be seen in Figure 6, there is a tendency of a near-linear latency increase within each packet rate group (of both sets) with the increase of the number of accommodated flows. This can be attributed to the increase of traffic load over time. Figure 7 shows the per-packet delay distribution of a sample flow from the first set of group rates with the mean (μ) and standard deviation (σ_D) values. The results show that per packet delay variation (with $\sigma_D = \pm 0.079$ ms) of a flow, served by the hardware table, is not experiencing significant fluctuations over time and remains relatively stable. This is a behavior that meets the expectations of a flow installed in the hardware table.

With regards to the *flow migration scenario*, the distributions of the average per-flow delays of the flows from the first set of packet rate groups (10-20-30 pps) for both scenarios (baseline and migration) are compared in Figure 8. There are 44 migrated flows that now experience higher average delays, since they are served (for a portion of their lifecycle) by the software table, as compared to the other flows, which were not affected by the migration process and were served only in hardware. In this experimental setting with predefined parameters (e.g., the total number of flows defined, the number of flows to consider for migration in each iteration, the capacity thresholds of the flow tables), the migrated flows belong only to the lowest 10 pps group since the accumulated pps rate of the group (500 pps) exceeded the software threshold limit (400 pps). For the remaining flows the impact is similar to the baseline scenario. This indicates that the placement algorithm does not affect the performance of the non-migrated flows. This is also confirmed in Figure 9, where the per-packet latency distribution of a sample (non-migrated) flow does not change significantly over time and is comparable to the baseline case in Figure 7.

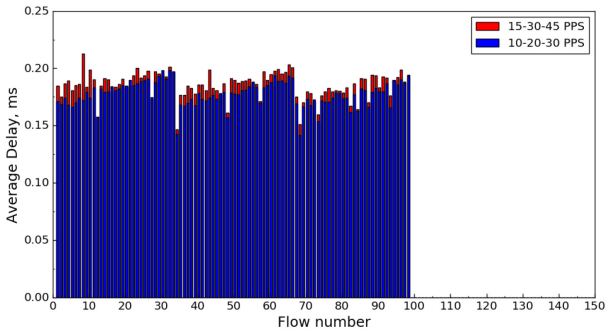

Figure 6: Baseline scenario. Average per-flow delay for 2 rate group sets

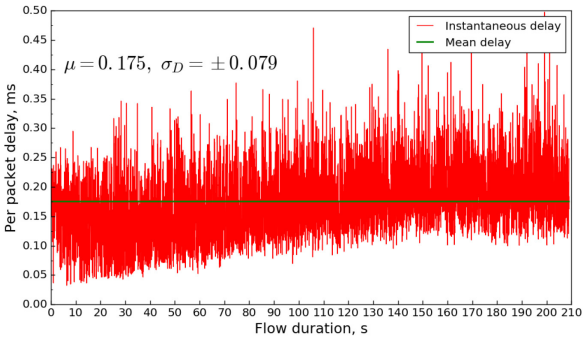

Figure 7: Baseline scenario. Per-packet delay of a sample flow

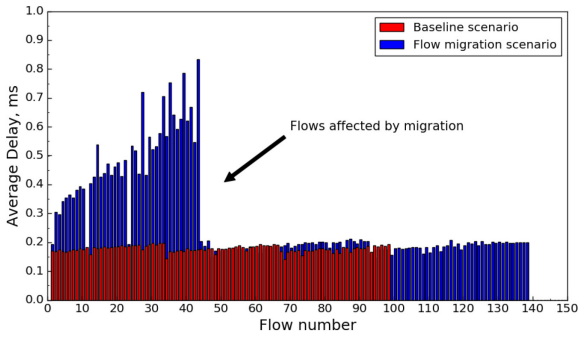

Figure 8: Baseline vs Flow migration scenario. Rate group set: 10-20-30 pps

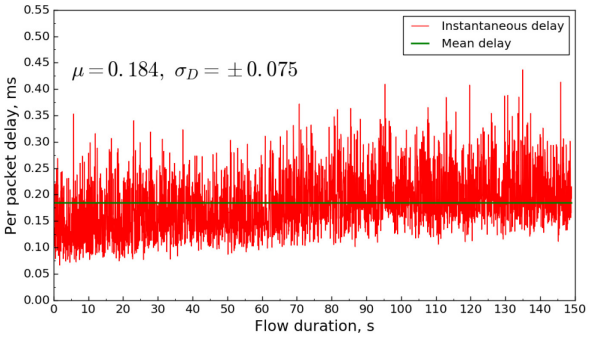

Figure 9: Per-packet delay of a sample non-migrated flow (migration enabled)

Considering the total number of accommodated flows, it is increased to 138 when migration is enabled, as compared to 99 in the *baseline scenario*, while the remaining flows were rejected as expected, because the capacities of both flow tables were fully utilized.

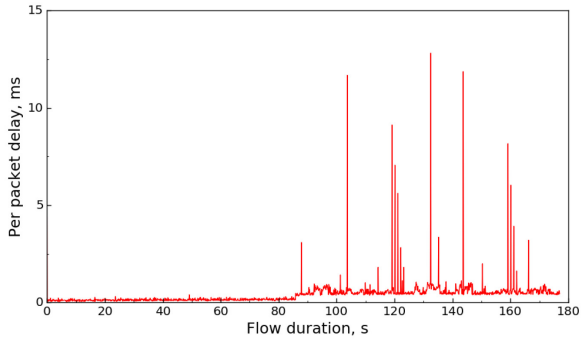

Figure 10: Per-packet delay of a migrated flow (10-20-30 pps set)

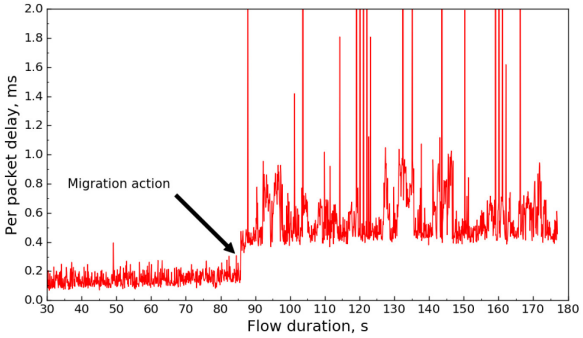

Figure 11: Per-packet delay of a migrated flow (a fragment of Figure 10)

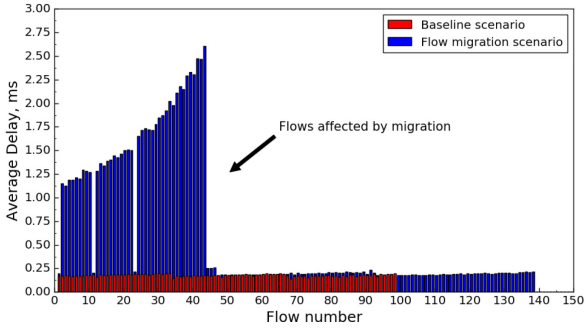

Figure 12: Baseline vs Flow migration scenario. Rate group set: 15-30-45 pps

Another aspect is how the packet processing delay changes when a flow rule is migrated. Figure 10 presents the evolution of per-packet processing delay of a sample migrated flow, and Figure 11 shows a zoomed-in fragment of it. We can observe a sharp increase of latency (the migration point in Figure 11) after the migration process is completed, since the processing is handled in the software table from there on. The impact of software processing is clearly seen in the form of stochastic latency spikes that can be a result of having shared interrupt-based processing in the CPU (Central Processing Unit) and memory buffer resources of the switch. The delay evolution pattern of all the migrated flows of this set of group rates is identical, with a sharp latency jump, higher delay variance and spikes after the migration.

The per-flow delay measurement results for the flows from the second set of packet rate groups (15-30-45 pps) for both scenarios (baseline and flow migration) are depicted in Figure 12. The distribution of the average delay of the non-migrated flows has identical pattern as compared to the baseline case. The increase of per-flow-group packet rates by ~ 33.33% resulted in a corresponding threefold increase of the average per-flow delays, and the increase pattern is observed to be non-linear.

The latency evolution of the individual migrated flows from this rate group set indicates that there is a significantly larger density of latency spikes with an area of excessive high magnitude spikes, reaching up to 100 ms. This behavior is presented in Figure 13 and its enlarged fragment in Figure 14. Such packet processing effects were observed in all the migrated flows and appeared at relatively the same (universal) points in time; these results indicate that larger number of packets are experiencing performance degradation due to higher load on the CPU-based subsystem of the switch.

It is important to emphasize that the observed spikes in delay appear after all the flows have been installed in the switch by the SDNC (in both the hardware and software tables), at which point the SDNC was not issuing any flow-rule-related actions in the switch. Thus, this behavior is purely associated with the switch, and not with the SDNC and/or the implemented placement algorithm.

The observed general variation of the measured delay, present in both scenarios, can be a consequence of the inherent hardware processing effects, e.g., clock drift and clock skew of the traffic generation server, affecting the packet timestamping accuracy, and internal memory buffer limits as well as packet queueing delays in the SDN switch.

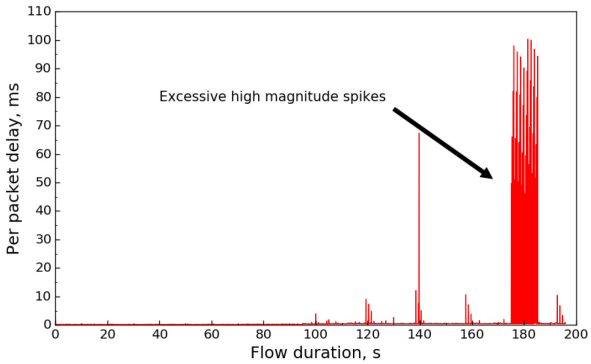

Figure 13: Per-packet delay of a migrated flow (15-30-45 pps set)

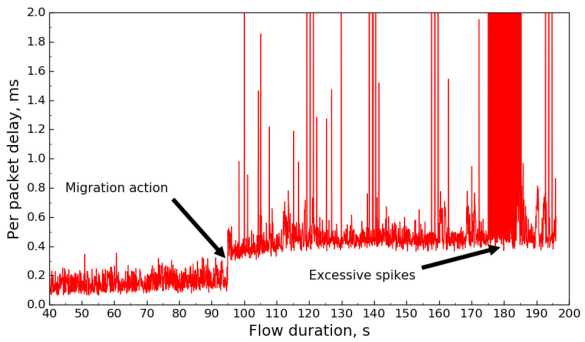

Figure 14: Per-packet delay of a migrated flow (a fragment of Figure 13)

As illustrated in Figure 8 and Figure 12, there is a trend of a sequential increase in average delays for the migrated flows, as more and more flows are accommodated. During the

migration process the flow rules, to be migrated, are retrieved by means of SDNC-specific functionality, without the opportunity to order them in a custom way. Assuming that the last flow installed in the hardware table of the switch is the first flow retrieved from the list provided by the SDNC, the trend shows that the flows that spent most of the time in the hardware table are experiencing the lowest average delays. This is a behavior that follows the performance characteristics of the two table implementations. Finally, we compared the distributions of the per-flow packet loss in both scenarios, and the results show that no packet loss was experienced.

It is important to emphasize that, since we had to virtually cap the processing capacity limits of the hardware and software tables, due to the limitations of our testbed setup (traffic server), we were not able to reach the effective (maximum) processing capacity limits of these tables. Therefore, if the real processing limits would be reached, the results could evolve in a different (non-linear) way, and the performance trends presented in this work, would have to be adjusted accordingly. However, even with virtual capacities, a clear trend was observed in the performance characteristics of the software table implementation. To obtain more indicative results, we need a more accurate traffic generation and measurement mechanism to be able to find the optimal values of the table performance settings, e.g., DPDK-based (Data Plane Development Kit) [16] solution.

VI. Conclussions

This work presented a flow-rule placement algorithm for SDN switches with hybrid flow table implementations. The algorithm is designed to utilize the flow rule capacities of both hardware and software tables, whilst also taking into account their inherent characteristics and limitations. The algorithm was implemented for the ONOS SDN controller and validated/evaluated on a physical SDN testbed. The results indicate that using the placement algorithm allows accommodating a larger number of flows, while limiting the degradation in network performance for the migrated flows and without impacting the non-migrated flows. Apart from that, the algorithm does not incur any packet loss. The downside is stochastic delay spikes affecting the migrated flows, which are caused by the inherent limitations of software-based processing of the switch. Since the behavior of the software table heavily depends on the flow packet rates we believe that the use-case of the algorithm could be to offload low pps low priority flows to the software table. However, for the algorithm to be able to perform, the switches, on which it is going to be enforced, must first be evaluated in terms of their software table performance, so that the appropriate thresholds can be set. Finally, a set of future work proposals is outlined as follows: 1) It might be of interest to model the performance of the software table, with regards to its utilization. The results can then be used as feedback on the placement decisions; 2) for a non-PoC implementation of the algorithm, the per-flow packet rates should be measured using a dynamic and accurate channel (e.g. sFlow); 3) more accurate traffic generation and measurement means should be used to be able to perform stress-testing of the SDN devices and the developed algorithm.

Acknowledgment

This work has been performed in the framework of the NGPaaS project, funded by the European Commission under the Horizon 2020 and 5G-PPP Phase2 programmes, under Grant Agreement No. 761 557 (http://ngpaas.eu).

References

[1] K. Pagiamtzis, S. Member, A. Sheikholeslami, and S. Member, “Content-Addressable Memory (CAM) Circuits and Architectures : A Tutorial and Survey,” vol. 41, no. 3, pp. 712-727, Feb. 2006.

[2] A. Mimidis, C. Caba, and J. Soler, “Dynamic aggregation of traffic flows in SDN: Applied to backhaul networks,” In *Proc. IEEE NetSoft Conf. Work. Software-Defined Infrastruct. Networks, Clouds, IoT Serv.*, Seoul, pp. 136-140, 2016.

[3] S. Das *et al.*, “Application-aware aggregation and traffic engineering in a converged packet-circuit network,” in *Proc. Opt. Fib. Comm. Conf. and Exposition and the National Fib. Optic Eng. Conf.*, Los Angeles, pp. 1-3, 2011.

[4] M. Rifai *et al.*, “MINNIE: An SDN world with few compressed forwarding rules,” *Comput. Net.*, vol. 121, pp 185-207, July 2017.

[5] S. Luo, H. Yu, and L. M. Li, “Fast incremental flow table aggregation in SDN,” in *Proc. Int. Conf. Comput. Commun. Networks, ICCCN*, Shanghai, pp. 1-8, 2014.

[6] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, “Infinite CacheFlow in Software-Defined Networks,” in *Proc. HotSDN Workshop*, Chicago, pp. 175-180, 2014.

[7] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, “CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks,” in *Proc. Symp. on SDN Research*, Santa Clara, pp. 1-12, 2016.

[8] X. Li and W. Xie, “CRAFT : A Cache Reduction Architecture for Flow Tables in Software-Defined Networks,” in *Proc. IEEE Symp. on Computers and Comm. (ISCC)*, Heraklion, pp. 1-6, 2017.

[9] Z. Guo *et al.*, “JumpFlow: Reducing flow table usage in software-defined networks,” *Comput. Net.*, vol. 92, Part 2, pp. 300-315, 2015.

[10] J. F. Huang, G. Y. Chang, C. F. Wang, and C. H. Lin, “Heterogeneous Flow Table Distribution in Software-Defined Networks,” *IEEE Trans. Emerg. Top. Comput.*, vol. 4, no. 2, pp. 252 -261, July 2016.

[11] A. Marsico, R. Doriguzzi-Corin, and D. Siracusa, “Overcoming the memory limits of network devices in SDN-enabled data centers,” *in Proc. Symp. Integr. Netw. Serv. Manag.*, Lisbon, pp. 897-898, 2017.

[12] H. Li, S. Guo, C. Wu, and J. Li, “FDRC: Flow-driven rule caching optimization in software defined networking,” in Proc. *IEEE Int. Conf. Commun.*, London, pp. 5777-5782, 2015.

[13] “Open vSwitch.” [Online]. Available: http://openvswitch.org/.

[14] O. N. Foundation, “OpenFlow Switch Specification,” pp. 1-177, 2015.

[15] “ONOS.” [Online]. Available: http://onosproject.org/.

[16] "Data Plane Development Kit," [Online]. Available: http://dpdk.org/